2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture

DRAF: A Low-Power DRAM-based Reconfigurable Acceleration Fabric

Mingyu Gao[†] Christina Delimitrou[†¶] Dimin Niu[§] Krishna T. Malladi[§] Hongzhong Zheng[§]
Bob Brennan[§] Christos Kozyrakis[†‡]

Stanford University[†] Samsung Semiconductor Inc.[§] Cornell University[¶] EPFL[‡]

{mgao12, cdel, kozyraki}@stanford.edu
{dimin.niu, k.tej, hz.zheng, bob.brennan}@ssi.samsung.com

***Abstract*—FPGAs are a popular target for application-specific accelerators because they lead to a good balance between flexibility and energy efficiency. However, FPGA lookup tables introduce significant area and power overheads, making it difficult to use FPGA devices in environments with tight cost and power constraints. This is the case for datacenter servers, where a modestly-sized FPGA cannot accommodate the large number of diverse accelerators that datacenter applications need.**

This paper introduces DRAF, an architecture for bit-level reconfigurable logic that uses DRAM subarrays to implement dense lookup tables. DRAF overlaps DRAM operations like bitline precharge and charge restoration with routing within the reconfigurable routing fabric to minimize the impact of DRAM latency. It also supports multiple configuration contexts that can be used to quickly switch between different accelerators with minimal latency. Overall, DRAF trades off some of the performance of FPGAs for significant gains in area and power. DRAF improves area density by 10x over FPGAs and power consumption by more than 3x, enabling DRAF to satisfy demanding applications within strict power and cost constraints. While accelerators mapped to DRAF are 2-3x slower than those in FPGAs, they still deliver a 13x speedup and an 11x reduction in power consumption over a Xeon core for a wide range of datacenter tasks, including analytics and interactive services like speech recognition.

***Keywords*-DRAM; reconfigurable logic; FPGA; low-power**

I. Introduction

The end of Dennard scaling is motivating the use of application or domain specific accelerators as an energy efficient way to improve performance [1]–[3]. For high volume and relatively stable applications, it is cost effective to design accelerators using full-custom (ASIC) techniques. For instance, all smartphone chips integrate programmable cores with specialized accelerators for video and image transcoding. For most other applications, it is better to map the accelerators on a configurable substrate like field programmable gate arrays (FPGAs) to achieve a balance between flexibility (programmability) and energy efficiency [4], [5]. The flexibility and efficiency of FPGAs have recently enabled a number of accelerators for datacenter applications, a domain where servers are shared between multiple types of applications and key algorithms change quite often [6]–[9].

FPGAs can realize arbitrary logic functions through the use of lookup tables (LUTs) linked with an interconnect, both of which are configurable at the bit-level [10]. Unfortunately, this approach has significant area and power overheads, making it difficult to use FPGAs in constrained environments. For example, datacenter servers are optimized for total cost of ownership (TCO) and have tight budgets for power consumption and cooling [11]. This makes it problematic to deploy large, power-hungry FPGA devices that have sufficient logic capacity to support powerful accelerators for multiple tasks and applications [6].

This paper presents a *DRAM-based Reconfigurable Acceleration Fabric* (DRAF), a substrate for bit-level reconfigurable logic that trades off some of the performance of FPGA devices for major improvements in area efficiency and power consumption. DRAF uses commodity DRAM technology to implement dense LUTs with wide inputs and outputs (e.g., 6 to 9-bit inputs, 2 to 4-bit outputs). Moreover, each LUT can store multiple contexts, allowing for several accelerator designs to be stored within a DRAF device. Applications can switch between accelerators without the overhead of reconfiguring an FPGA from off-chip memory.

While the use of DRAM technology provides density and power efficiency, it also introduces significant latency overheads compared to SRAM-based FPGAs. We carefully engineer DRAF arrays and their timing to overlap the DRAM latency with the latency of the interconnect fabric. We also implement a timing scheme that avoids the loss of design data due to the destructive nature of DRAM reads as signals propagate through the reconfigurable fabric.

We compare DRAF to FPGA devices and programmable cores across a wide set of accelerator designs for datacenter workloads, including machine learning and analytics. An 8-context DRAF array requires 19% lower area than a single-context FPGA array, which represents a 10x improvement in area efficiency. DRAF consumes less than 1/3 of the power of an FPGA array. For the accelerators examined, the maximum power (TDP) of DRAF is 3.57 W, compared to 15.89 W for an FPGA. While the clock frequency of accelerators mapped to DRAF is 2-3x lower than when mapped to FPGAs, DRAF is still able to achieve a 13x

1063-6897/16 $31.00 © 2016 IEEE
DOI 10.1109/ISCA.2016.51

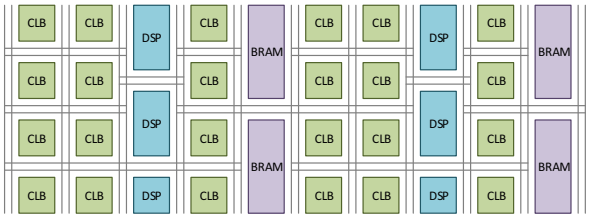


Figure 1. The FPGA array organization. Block sizes and numbers may vary in each specific device.

average speedup at 11x lower power consumption over a high-end, programmable core. Even if we assume ideal multi-core scaling, DRAF provides a significant energy advantage over programmable cores. Overall, these results establish DRAF as an attractive and flexible acceleration fabric in cost (area) and power constrained environments.

II. Background

A. Reconfigurable Logic and FPGAs

By leveraging spatial programmability to target application-specific optimizations, reconfigurable fabrics can provide significant performance and energy efficiency improvements over programmable cores. Moreover, the post-fabrication reconfigurability greatly shortens the long development time of ASIC hardware and allows specialization for low volume applications. These advantages make reconfigurable fabrics the most popular target for accelerator designs for numerous domains, including image processing, financial analysis, security, bioengineering, and network processing [4].

FPGAs represent the most successful and widely used reconfigurable fabric thus far. FPGAs use lookup tables (LUTs) and flip-flops (FFs) as the basic primitives. A LUT is a small memory unit that can realize an arbitrary combinational logic function by storing all potential output values. Given a set of input bits as the address, the LUT returns the corresponding output value. Modern FPGAs typically use LUTs with 6-bit inputs and 1-bit outputs, with the ability to be split (i.e., *fracture*) into multiple smaller ones, as well as special support for carry logic. A LUT is usually implemented with an SRAM array or with individual SRAM cells and a multiplexer tree. The FFs allow for data retiming and storage for sequential logic. Multiple LUTs and FFs are grouped into configurable logic blocks (CLBs). Modern FPGAs also have dedicated DSP blocks for multiplication and other complex operations, and block RAMs (BRAMs) for on-chip high-capacity data storage.

The different types of blocks in FPGAs are typically organized in a 2D grid using the "island layout" shown in Figure 1 [12]. The interconnect between blocks contains wire segments (called *tracks*) with different lengths, and configurable multiplexers and switch boxes (not shown in the figure). The number of tracks is pre-set during fabrication, usually on the order of several hundreds for moderate and large FPGAs [12], [13]. By writing different configurations into the SRAM cells that control the multiplexer select bits, we can connect different sources (tracks or block output ports) to a certain sink (the input port of a block or another track). Note that the interconnect configuration in most FPGAs is static, i.e., decided at synthesis time, and never changes once loaded to the device [14], [15].

FPGAs in datacenters: FPGAs have recently been used to accelerate datacenter applications [6]–[9]. This is a challenging domain for FPGA use. First, it is difficult to introduce an accelerator device that consumes more than 10 to 20 W in a server without significant rework of its power provisioning and cooling design [6]. Second, datacenters are optimized for total cost of ownership (TCO), which makes it difficult to introduce expensive devices in each server. Third, datacenter servers often host multiple applications, each of which may need a separate accelerator. Finally, many datacenter workloads require fairly large accelerators that can consume a large number of FPGA primitives. The first two requirements push towards the use of small FPGAs that are cost and power efficient. The latter two requirements push towards the use of large FPGAs that can fit multiple, potentially large accelerators, without long reconfiguration delays as different applications frequently start and end on the server (high churn). One approach to manage this tradeoff is to use a medium-sized FPGA in each server with a separate, custom-designed interconnect between FPGAs that allows requests from each server to be served by several of these devices [6]. Our work aims to build a new reconfigurable device that provides a significant improvement in logic density and power consumption over existing FPGAs, addressing the tradeoff directly. Multiple such devices can still be linked together if needed.

Multi-context reconfigurable fabric: *Multi-context* FPGAs address the slow reconfiguration of switching between designs mapped on the fabric [16]–[18]. Each context supports the configuration for a separate accelerator design. Multi-context FPGAs store multiple configurations for LUTs and interconnect control on chip, and can rapidly switch between them at runtime. This is similar to coarse-grained thread switching in multi-threaded processors that can quickly switch between threads but at each point in time only a single thread uses the processor resources. Tabula's 3PLD is a recent commercial example of an FPGA with multi-context support [18].

The primary disadvantage of multi-context FPGAs is the space and time overheads of the additional configuration storage. Previous multi-context proposals stored the configurations in separate backup memory units, either in a per-LUT fashion [16] or together in a global configuration memory [17]. This backup memory occupied significant area that could be used to implement additional normal LUTs to

increase the capacity for a single context instead. Moreover, when switching between contexts, loading the configuration from backup memory to the logic blocks introduced non-negligible latency and power overheads. Since FPGAs have been mostly used for single accelerator or control logic design, multi-context FPGAs were largely abandoned by the industry and research community. Nevertheless, the use of FPGAs in shared datacenters makes multi-context FPGAs an attractive approach once again.

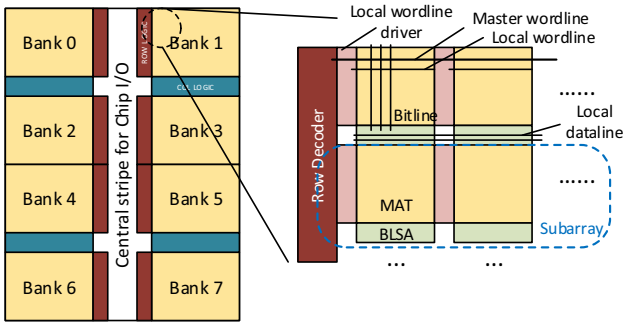
B. DRAM Basics

DRAM is used as the main memory technology in most computing systems. It offers high density (low cost per bit), relatively low access latency, and infinite endurance. In a typical memory system for DDRx DRAM, 8-16 memory chips with a narrow interface are packed into a wide interface, DIMM module, and are accessed in lock-step [19]. The processor chip connects to DIMMs through memory channels. Each channel is managed by an independent controller and can host a small number of DIMMs.

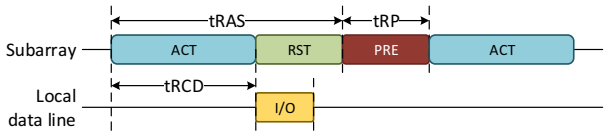
For this work, it is important to understand the internals of a single DRAM chip. The DRAF devices presented in Section III and IV are implemented using DRAM technology, but used as FPGA devices connected to the processor through a PCIe channel, not a memory channel. To maximize access parallelism, the memory cells are organized into multiple *banks*, as shown in Figure 2(a). Banks can operate independently but share the central I/O interface. Each bank is composed of smaller arrays called *MATs*. The width and height of a MAT are typically 512 to 1024 cells. A group of 8 to 16 MATs in a row, called the *subarray*, shares the row decoding logic[1]. The horizontal wordline output of the row decoder is hierarchical [24], [25]. The decoder directly drives only the *master wordlines* (routed in metal layers), and the local wordline drivers between MATs connect them to *local wordlines* (routed in gate poly) to the access transistor of each DRAM cell. This greatly reduces access latency; alternatively the load of a single wordline would be too large for fast signal propagation. The vertical bitlines are connected to the *bitline sense amplifiers* (BLSAs, or sense-amps). From there, output data is routed to the central chip I/O interface through local and global datalines.

DRAM subarray operation: A DRAM cell includes a capacitor to store the bit value (0 or 1) and an access transistor. The access is destructive and the cells need to be recharged/discharged afterwards. Figure 2(b) shows the timing of a DRAM subarray read with the chip-level commands and data bus states omitted [26], [27]. The decoded row address drives the master and local wordlines in the subarray which conduct the access transistors and allow DRAM cells

[1]DRAM terminology varies with respect to MATs and subarrays. Our terminology is consistent with [20], [21]. Others use subarray to refer to a basic tile, i.e., the MAT [22], [23].



(a) DRAM bank organization.



(b) Access timing for DDRx DRAM.

Figure 2. Layout and access timing for DDRx DRAM.

to share their charge (if any) with the bitlines. The sense-amps amplify the small voltage change in the bitlines to full swing signals of V_{DD} or 0. The charge sharing and sensing together are called *activating* (ACT).

When the bitline voltage has been amplified sufficiently, it is safe to read from the sense-amps without flipping their values. The selected columns are connected to the local datalines (I/O). In the background, the cell capacitors need to be fully recharged/discharged to *restore* (RST) to the original values. After the restoration, the cells are disconnected from the bitlines. Now, the bitlines and sense-amps can prepare for the next access by being *precharged* (PRE) to V_{ref}. Note that PRE does not need to follow right after RST. Under an *open-page* policy we can keep the data in the sense-amps in case the next access goes to the same row.

At the subarray level, the DRAM latencies depend heavily on its size [26], [27]. For smaller MATs, the bitline capacitance is small, making sensing and precharging fast, hence reducing the ACT and PRE latencies. The number of MATs per subarray has a smaller impact on timing due to the hierarchical wordline structure, but the local wordline drivers consume non-negligible area compared to the typical MAT size. At the chip level, the DRAM latency is also affected by the various parameters of the central I/O block and the command/data latencies of the memory channel.

C. Challenges for DRAM-Based FPGAs

The goal of this paper is to use DRAM subarrays to implement a new reconfigurable fabric with high logic density and low power consumption that also provides multi-context support. These features make this new fabric highly suitable for a number of constrained applications, including those hosted in shared datacenters.

Table I
COMPARISON OF DRAF TO CONVENTIONAL FPGAS.

	FPGA	DRAF
LUT technology	SRAM-based	DRAM-based
LUT delay	Short (sub-ns)	Long (ns)
LUT output width*	1-bit	Multi-bit
Logic capacity	Moderate	Very high
Configurations	Single	Multiple
Power consumption	Moderate	Low
Cost	High	Low

*: Without fracturing.

The naïve approach is to use the high-density DRAM subarrays to replace the SRAM LUTs in FPGAs. However, this simple substitution would run against several challenges. First, DRAM has much higher access latency than SRAM. Even for a single subarray without the chip-level delays, the access latency is about 10 ns. Using a smaller subarray can drop latency down to 2-3 ns, which is still 30x to 50x slower than a SRAM-based LUT [28]. Without careful optimization, DRAM-based reconfigurable logic can hardly provide any speedup over a programmable core. Second, DRAM accesses are destructive and require explicit activate and precharge operations. The timing of accesses to DRAM-based LUTs must be carefully managed to avoid destroying the LUT contents while its inputs are still not stable.

Finally, DRAM subarrays are optimized for area and cost efficiency. As a result, their capacity is fairly large (e.g., 8 Mbits), and their inputs and outputs are wide (e.g., 17-bit address and 64-bit data outputs). This is significantly larger than the capacity and the number of input/output bits of the typical FPGA LUT (6-bit input, 1-bit output). If we simply present each DRAM subarray as a single 17-bit input, 64-bit output LUT, most of these LUTs will be greatly underutilized when mapping real-world accelerators on them. Ideally, we want to use DRAM resources to support LUTs with a smaller number of input (6-10) and output bits (2-4), which can be highly utilized by most designs.

III. THE DRAF ARCHITECTURE

DRAF leverages DRAM technology to implement a reconfigurable fabric with higher logic capacity and lower power consumption than conventional FPGAs, such as the Altera Stratix and Xilinx Virtex series [29], [30]. DRAF supports multiple contexts with fast context switching, enabling lightweight device sharing across multiple accelerators for one or more applications. Table I summarizes the key features of DRAF. Since DRAF is optimized for low cost and power consumption, we can introduce accelerators without a significant increase in the overall TCO or power budget.

DRAF has several similarities to conventional FPGAs. It uses LUTs to support programmable logic but implements them with DRAM subarrays (see Section III-A and Section III-B). LUTs and flip-flops (FFs) are grouped into configurable logic blocks (CLBs). DRAF uses DSP blocks for complex arithmetic operations, as well as BRAM blocks for embedded memory implemented with DRAM instead of SRAM (Section III-C). For simplicity, DRAF also uses the common array layout shown in Figure 1. The interconnect between logic blocks is a statically configured fabric similar to those in FPGAs, but with low-cost time-multiplexing support (Section III-D). The rest of this section focuses on the design challenges specific to the use of DRAM and the DRAF features that are different from conventional FPGAs.

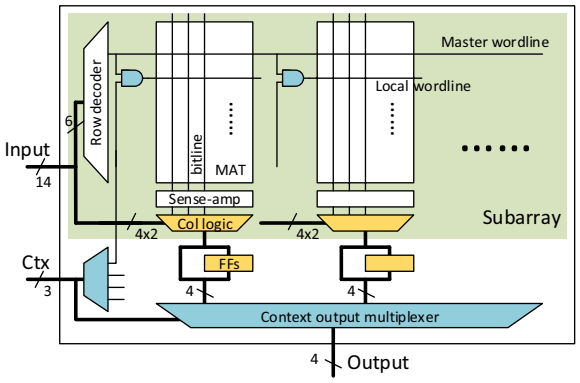


Figure 3. The basic logic element (BLE) in DRAF. As a typical example, the size of the LUT is 8-bit (6 + 2) input and 4-bit output, with 8 contexts.

A. The Basic Logic Element

Figure 3 shows the *basic logic element* (BLE) in DRAF that includes a multi-context LUT, one or more FFs, and auxiliary multiplexers. Similar to FPGAs, a CLB contains multiple independent BLEs with local interconnect. The DRAM subarray in the shaded portion of Figure 3 implements the LUT. The LUT inputs are fed as the row and column address to the DRAM subarray and a memory access produces the LUT output. The DRAF LUT has 6-10 input bits and 2-4 outputs bits, slightly wider than those in a conventional FPGAs. The density of DRAM subarrays allows a DRAF CLB to provide 10x logic capacity over an FPGA CLB of the same area. We study in depth LUT sizing in Section VI-A. The FFs in the BLE allow for the output signals to be optionally registered (retimed) for subsequent use in sequential logic. The different MATs in the subarray provide the configuration storage for different contexts.

In conventional DRAM design, the column logic multiplexes the sense-amps to the subarray outputs. For instance, this logic may pick the output of every 64th sense-amp. To improve logic flexibility for our multi-output LUT, we change the column logic to allow for each output bit to be independently selected from the corresponding set of sense-amps. In Figure 3 for example, the multiplexer for each of the 4 output bits in one MAT uses a separate 2-bit control signal. Each LUT output now uses a partially different set of the input signals (different column address). The additional

flexibility enables higher LUT utilization while amortizing the cost of the row decoder across all LUT outputs. It also allows us to easily support fracturable LUTs in DRAF, i.e., dividing the LUT into two or more smaller LUTs.

Multi-context support: In conventional DRAM, we concurrently access all MATs in a DRAM subarray and read a wide sequential output that is gradually transmitted off-chip. For reconfigurable logic, such a wide LUT output is practically useless, as we rarely need to calculate tens of functions on the same set of inputs. Moreover, even though we only care about 2-4 outputs from one MAT at a time, activating and amplifying a row across all MATs (8 Kbits) lead to substantial energy overheads.

To address these issues, we reduce the width of each MAT, and further decouple the MAT access in the subarray to limit each LUT to a single MAT access, which allows us to implement fine-grained functions with 6-10 inputs and 2-4 outputs. We use the remaining MATs for *multi-context* support. Each MAT implements the LUT for an independent FPGA configuration. Each configuration can support a different accelerator design. The cost of each configuration is sublinear as the MATs share the row decoding logic. All the input and output ports are also shared among all contexts.

We leverage the commonly-used hierarchical wordline structure of DRAM subarrays to implement the single MAT access [24], [25]. We add an `AND` gate to each local wordline driver to combine the master wordline signal with the context enable signal decoded from a global context counter (see Figure 3). This allows us to only activate and precharge a single MAT corresponding to the currently-selected context [23]. A *context output MUX* multiplexes the selected context to the output port. Note that the context output MUX is after the FFs. Storage logic, such as FFs, cannot be shared and must be replicated for each context to avoid one context overwriting other contexts' data [17].

Compared to previous multi-context FPGAs [16]–[18], the multi-context support in DRAF has several advantages. First, the area overheads due to the `AND` gates are negligible as hierarchical wordline structure is already used in DRAM. Second, there is little dynamic power overhead as we only access a single MAT. If some contexts are not used at all, we can fully power gate them to avoid static leakage. Third, in addition to storing all the contexts on chip, the configurations are already loaded in the LUTs. Context switches are as fast as a simple update to the context counter. The new context will be ready to use in the next clock cycle. There is no need to pause and load configurations from on-chip or off-chip structures, introducing no downtime for the upper-level applications. Hence, if we use multiple contexts to support multiple accelerators for one or more applications, we can quickly switch between accelerators on demand.

The multi-context BLE can also support inter-context chaining (also known as "temporal pipelining" [31]), where the BLE output for one context becomes the BLE input for another context. This allows for accelerator designs that exceed the capacity of a single context in a DRAF array. To support inter-context chaining, we must add a configurable crossbar between the LUT and FFs or make the select signal of the context output MUX fully configurable. Either approach introduces non-negligible area and energy overheads. Inter-context chaining also requires support from the CAD flow to properly split a design across contexts. Our current DRAF design does not support inter-context chaining; we leave the exploration of its implementation and benefits to future work.

In summary, the DRAF BLE supports multi-context reconfigurable logic with very fast switching time, low power overheads, and high area efficiency.

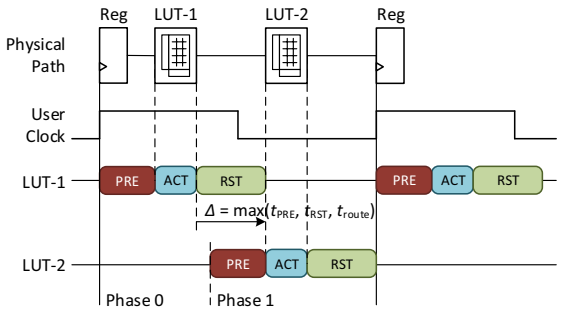


Figure 4. Timing diagram of DRAF LUT operations, where LUT-1 produces an signal used as the input of LUT-2. Δ is the overlapping of restoration, precharging, and routing delays.

B. Timing Optimization

Accessing a DRAM subarray to read data or evaluate a LUT function requires explicit precharge, activation, and subsequent restoration due to the destructive nature of DRAM accesses. This makes our DRAM-based LUT significantly slower than the conventional SRAM-based LUT. We must also be careful with ordering events in the DRAM-based LUT. Specifically, each LUT must evaluate its function (i.e., activate its subarray) only after all its input signals are valid (i.e., all previous LUTs have safely generated these signals). This is not an issue with the SRAM-based LUTs since SRAM reads are not destructive.

To manage timing and ordering constraints, DRAF divides each user design cycle into multiple phases[2]. We assign a specific phase for the activation of each LUT in the design, as shown in Figure 4. The phase of a LUT must be greater than the phase of all LUTs producing its input signals. If all the inputs are from registers (FFs), its phase is 0. We delay the precharge needed for the next user cycle until right before the next activation (similar to open-page policy)

[2] A user cycle is the clock cycle at which the design mapped on DRAF operates. It is an integer multiple of the internal clock cycle at which the DRAM peripheral logic operates.

to ensure that the LUT outputs are valid for the remainder of the current cycle. This is necessary if the inputs of a LUT come from different phases. Each LUT has a simple logic shared across all contexts that executes the fixed PRE-ACT-RST sequence on its subarray according to the phase configuration. No commands are transferred across the chip. Given these constraints, DRAF LUTs can be chained to evaluate arbitrary large logic functions in each user cycle.

The phase for each LUT is part of the configuration stored in each LUT, and is statically assigned by CAD tools during synthesis and mapping. The assignment can be combined with the algorithm that finds the critical path. The number of phases in the cycle is equal to the number of DRAF LUTs on the critical path. Hence, to minimize the user clock cycle, we should minimize the number of phases.

We hide the timing overheads of DRAM restoration and precharging by overlapping them with the routing delay within the DRAF array. Specifically, we overlap the restoration of a LUT that produces a signal, with the time for precharging the destination LUT of this signal, and the time for routing this signal between the two LUTs, as shown in Figure 4. This three-way overlapping is possible because the output of a LUT is preserved during restoration, and no stable inputs are needed for a LUT during precharging.

Overlapping routing delay with DRAM timing overheads is a critical optimization for DRAF. The raw access delay of a DRAF LUT is 30x to 50x longer than the delay of a SRAM LUT (see Section II-C). However, routing latency is typically the dominant user cycle component in conventional FPGAs [32]. Hence, the overlapping allows DRAF's critical path delay to be only 2x to 4x longer than that of conventional FPGAs. Hence, DRAF provides reasonable performance while achieving significant improvements in power consumption and density (see Section VI).

DRAM sense amplifiers as user registers: In the DRAF BLE, we could potentially eliminate the separate FFs, which are replicated for each context and introduce significant area and energy overheads [17]. Instead, we could use the sense-amps, which already latch the output data and maintain them until the subarray is precharged in the next user cycle.

Using the sense-amps as user registers introduces two timing constraints. First, the phase of a LUT that consumes the registered data in the next user cycle must be smaller than the phase of the LUT that produces the data. Otherwise, the data will be lost due to precharge before the next LUT consumes it. Second, all outputs of a LUT must be managed with the same `EN` signals. In contrast, the separate FF for each output can be managed independently. Since it is not always feasible to fulfill these constraints, we opt to maintain the separate FFs in DRAF, but use them selectively only when needed. If these two constraints are met, we can bypass the FFs and eliminate their power overheads. Note that, for proper use of the sense-amps as user registers, we also preserve the column address necessary to select the proper subset of these bits in the subsequent user cycle.

DRAM refresh: Since cell capacitors leak, DRAM requires periodical refresh. We refresh all subarrays on a DRAF chip concurrently using a shared row address counter in each CLB and BRAM block. Concurrent refresh is already practical in terms of power consumption in commodity DRAM [33], [34] and much easier with DRAF as the subarray is much smaller. All used contexts are refreshed simultaneously, but we skip refreshing unused LUT contexts and BRAM blocks. We discuss the thermal impact of refresh in Section VI-C. Since we refresh all blocks in parallel and subarrays are small (see Section VI-A), the time required for refresh is no more than 256 ACT+PRE cycles. This translates to roughly 1 μs refresh pause every 64 ms. This is negligible overhead for both throughput and latency critical applications in datacenters.

Refresh is coordinated by the DRAF device driver. During refresh, the driver pauses operations on DRAF in a manner similar to how one stalls a processor pipelines. The driver does not provide new requests or inputs to the device and ignores any output, as if the device is busy. Within the device, the internal FFs hold the current data until refresh finishes. Since all the registers are preserved, the states of the system before and after refresh are identical. Note that if a user register uses DRAM sense-amps as described before, the data must be stored into the normal FFs before refresh and restored after, handled by the device driver as extra operations with the refresh commands.

C. Other Logic Blocks

In addition to CLBs that contain LUTs and FFs, the DRAF array includes dedicated arithmetic blocks and embedded memory blocks. All the logic blocks in DRAF are implemented in DRAM technology. Future designs could utilize 3D integration technology to tightly integrate blocks implemented in different processes.

The DSP blocks with dedicated multipliers support efficient, wide-word multiplication operations commonly-present in accelerator designs. We used a 25×18 multiplier similar to the DSP48E1 slice in Virtex-6 FPGAs [28]. Since DRAF uses DRAM technology, the multiplier blocks are slower compared to those in conventional FPGAs. Nevertheless, the multiplier latency is not critical as the dominant factor for DRAF performance is the LUT access latency. The area overhead of multiplier blocks is low since they do not need to be replicated across contexts.

Conventional FPGAs use SRAM arrays for memory blocks. For DRAF, it is natural to use DRAM arrays to implement high density memory. Their structure is similar to DRAM-based LUT, but with different input and output widths. We use 36 Kbit BRAM blocks with configurable input and output widths to provide identical functionality to the RAMB36E1 blocks in Virtex-6 FPGAs [28]. Due to the larger capacity, the RAM block has slightly higher delay

than the DRAF LUT, but the gap is much smaller than that for conventional FPGAs which can be more than 20x. Note that, since RAM blocks provide data storage, we need to share their capacity across all active contexts. Nevertheless, we do not need to sacrifice a higher percentage of area for RAM blocks in DRAF compared to conventional FPGAs. First, DRAM cells have a much higher density than SRAM cells. Second, we can minimize memory requirements by carefully sharing buffer space between accelerators [35].

D. Interconnect Fabric

DRAF uses a fully static interconnect fabric as it is the case with conventional FPGAs. Similarly to the logic blocks, the interconnect supports multiple contexts. Since the CLBs are DRAM-based and not particularly fast, it is sufficient to use a simple and static, time-multiplexing scheme to share the DRAF routing across contexts [14]. We replace each routing configuration bit in the connection boxes and switches with multiple bits, one for each context, using SRAM cells. Context switches activate the proper configuration bits in the fabric.

Since DRAF LUTs support wider inputs and outputs than FPGA LUTs, DRAF can pack more logic into each CLB and result in fewer CLBs. This reduces the pressure on global routing resources. Hence, the number of routing tracks needed for DRAF is smaller than that in FPGAs, which leads to further area savings as the interconnect accounts for a significant portion of area in FPGAs (see Section VI-B).

Note that the DRAF time-multiplexed interconnect is different from the proposed network-on-chip-based interconnects for FPGAs that use dynamic packet switching and overlay networks [36], [37]. The configuration of the interconnect in DRAF is statically determined and loaded into the fabric. There are no routing decisions during runtime. The static approach leads to significant power savings over packet-switched networks [15]. While there may be other benefits from using NoC-based interconnects in DRAF, we leave this topic to future work.

IV. System Design with DRAF

A. DRAF Design Flow

Since DRAF has the same primitives (LUTs, FFs, multipliers, BRAMS) as modern FPGAs, its design flow is very similar to that of FPGAs. Designers can develop accelerators using a hardware description language, such as Verilog or VHDL, or a high-level synthesis flow, such as the Xilinx Vivado HLS and the Altera OpenCL SDK [38], [39]. The CAD flow for these languages can process the design (code elaborate, synthesize, pack, place, and route) and map it to a DRAF device.

However, since DRAF differs in key architectural and timing characteristics, CAD tools need to be tuned to produce optimized designs for it. First, DRAF implements LUTs with wider inputs and outputs. The packing algorithm needs to pack more logic per LUT and to use the wider inputs and outputs. Second, because the logic density of DRAF is much higher than that of an FPGA, area is not a scarce resource anymore. In contrast, latency is a much more important commodity and should be the primary focus of optimization. Nevertheless, routing delay is easier to deal with as it overlaps with DRAM operations (see Section III-B). Third, CAD tools need to ensure that all the timing requirements of DRAF are satisfied after synthesis and encode the additional information, such as the phase for each LUT, into the configuration stream. As previously mentioned, this algorithm is similar to critical path finding. Finally, the CAD tools should be aware of the multi-context features, which is described below.

In this study, we use open-source synthesis tools that allow us to map Verilog-based designs to different reconfigurable fabrics [13], [40]. See Section V for the details.

B. Multi-context Operation

Similar to prior multi-context FPGAs [17], DRAF supports multiple operation modes. In the *independent* mode, the different contexts are completely independent of each other and implement separate accelerators for one or more applications in the system. When an accelerator is needed, it is enabled by updating the global context counter for the CLBs and the DRAF interconnect. The desired accelerator becomes instantly available to use. In this mode, a DRAF device can be virtually viewed as multiple FPGA instances that have no connection between them, but need to be used in a time-shared fashion. The CAD tools can also synthesize each context independently, and combine the configuration streams before loading them into DRAF.

The only important issue about the independent mode is BRAM space management between designs. If context switches happen in a coarse-grained manner and no data from the previous accelerator is left on the DRAF device, each context can assume that it has access to all BRAM resources. In contrast, if switches happen in a fine-grained manner and multiple accelerators store data in the device at any point in time, it is best to statically partition the available space. This can be achieved by embedding the context counter value in the higher bits of the address ports. Since the interconnect is statically routed, there is no security concern about one accelerator accessing the BRAMs assigned to another design.

Alternatively, a DRAF design can operate in the *partitioned* mode, where a large design is split into multiple partitions, each mapped to a different context (see inter-context chaining in Section III-A). The partitioned mode is quite powerful but much more complex to manage. The CAD tools need to automatically split the design into partitions and figure out the data flow between them. A runtime scheduler is also required to switch between partitions when signals flow from one partition to the next one. The Tabula

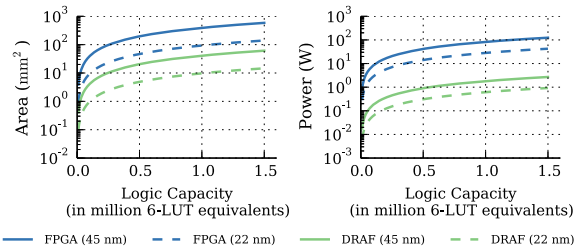


Figure 5. Area and maximum power comparison of FPGA and DRAF devices with different logic capacities for 45 nm and 22 nm technologies.

3PLD [18] is a commercial system that supports this mode. The partitioned mode can be combined with the independent mode. For instance, one large design may take 4 contexts while four other designs use the remaining 4 contexts.

Even though inter-context chaining and the partitioned mode are interesting, they are challenging to implement. Therefore in this paper we mostly focus on the independent mode and leave the partitioned mode to future work. As we will see in Section IV-D, the independent mode provides a shared fabric that can consolidate multiple accelerator designs used by different applications.

C. Devices & System Integration

A DRAF device consists of CLBs, DSP blocks, and BRAM blocks, as well as other peripheral and interface components. Similar to conventional FPGAs, we can create DRAF devices of various sizes. Figure 5 compares the area and maximum power consumption of DRAF and FPGA devices with different logic capacities measured in 6-LUT equivalents for 45 nm technology and also scaled down to 22 nm. See Section V for the detailed methodology. For a fixed logic capacity, DRAF devices provide a more than 10x area improvement and roughly a 50x power consumption improvement (the logic capacity accounts for all eight contexts in DRAF, but only one context can be used at a time). If we target a cost effective device size of 75 mm^2 at 45 nm, an FPGA can pack roughly 200k LUTs (as is the case for Virtex-6 FPGAs [28]). On the other hand, DRAF can store more than 1.5 million LUTs, a logic capacity comparable to that of the state-of-the-art Virtex-UltraScale+ FPGAs that use the much more recent 16 nm technology [41]. The power consumption advantage is also remarkable. While the FPGA power can easily exceed 10 W, DRAF only consumes about 1 or 2 W at very high capacity.

The most straightforward way to integrate a DRAF device into a computer is by directly connecting it to a multi-core processor through a high performance I/O link like PCIe. Alternatively, DRAF devices can be integrated to a processor chip using 2.5D and 3D integration. The low power consumption of DRAF is an advantage in this case, as heat removal is a major challenge for 3D integration. Moreover, DRAF devices can be 3D integrated with ordinary memory chips to support near-data processing (NDP) [42]. Reconfigurable logic is a promising substrate for NDP logic as it can provide the high processing throughput needed in a 3D stack, while maintaining programmability [43], [44]. DRAF is a particularly promising candidate for the logic layer of NDP stack. First, the high logic density allows DRAF to offer adequate processing functionality within the small logic die area to take the best advantage of the high memory bandwidth and parallelism. Second, DRAF has lower power consumption compared to traditional reconfigurable logic fabrics, thus can better satisfy the thermal constraints in 3D stacking. Third, the low clock frequency of DRAF due to longer LUT access latency is less critical in an NDP scenario, since the data access latency from DRAM dies is the major performance bottleneck. Processing throughput is the key requirement and can be achieved through high logic density and parallelism.

D. Using DRAF in Datacenter Servers

DRAF trades off some of the potential performance of FPGAs to achieve high logic density, multiple contexts, and low power consumption. These features make DRAF devices appropriate for both mobile and server applications, where one wants to introduce an FPGA device for acceleration without significantly impacting the power budget, airflow and cooling constraints of existing systems [45]. For mobile devices, it is difficult to accommodate an accelerator that consumes more than 0.1 to 1 W. For server systems, it is difficult to accommodate an accelerator that consumes more than 10 to 20 W [6], which is the typical power consumption of many PCIe-based devices like NIC and SSD controllers. The lower the power of a high density reconfigurable fabric is, the easier it is to introduce in systems of all types.

In datacenters that host public and private clouds, servers are routinely shared by multiple, diverse applications to increase overall utilization. Different applications and different portions of each application (e.g., RPC communication vs. security vs. main algorithm) require different accelerators. For example, a server may be shared by text analytics and video decoding tasks [7]. Both can benefit from application-specific accelerators. To serve a large number of applications with a conventional FPGA, we either need a large, expensive, and power-hungry FPGA that can fit all accelerators or use a small FPGA and deal with the latency of reconfiguration. The application downtime during the FPGA's reconfiguration is an additional vulnerability in system resilience [6]. In contrast, DRAF provides a shared fabric that supports multiple accelerators using different contexts. The high logic density ensures that each individual context has sufficient capacity for the different accelerator designs. Switching between the designs has little overheads in timing or energy. It also introduces no downtime since the new context is instantly ready to use in the next cycle.

Table II
THE ACCELERATOR DESIGN BENCHMARKS.

Number	Name	Source	Domain
1	aes	MachSuite [46]	Cryptography
2	backprop	MachSuite [46]	Neural network
3	bfs	MachSuite [46]	Graph
4	fast	Vivado HLS [38]	Image/video
5	gemm	MachSuite [46]	Dense linear algebra
6	gmm	Sirius [7]	Voice processing
7	harris	Vivado HLS [38]	Image/video
8	kmp	MachSuite [46]	Text processing
9	mergesort	MachSuite [46]	Analytics utility
10	spmv	MachSuite [46]	Sparse linear algebra
11	stemmer	Sirius [7]	Text processing
12	stencil	MachSuite [46]	Image/video
13	viterbi	MachSuite [46]	Voice processing
14	blobmerge	VTR [13]	Image/video
15	editdist	V. Kundeti [47]	Text processing
16	openrisc	VTR [13]	Soft core
17	sha	VTR [13]	Cryptography
18	vision	VTR [13]	Image/video

V. METHODOLOGY

Workloads: We evaluate DRAF as a reconfigurable fabric for datacenter applications by selecting a wide set of accelerator designs from several benchmark suites, including MachSuite [46], Sirius [7], Vivado HLS Video Library [38], and VTR benchmark suite [13]. These designs, summarized in Table II, capture a representative set of performance-critical kernels commonly used in large-scale production services which are not currently publicly available [6]–[9]. Most of them apply arithmetically intensive algorithms on streams of data and several of them include non-trivial control flows. The first 13 benchmarks have corresponding C/C++ implementations so we also evaluate their performance against CPUs in Section VI-C.

CAD tools: The benchmarks from [46] and [7] are written in C/C++. We use Xilinx Vivado High-Level Synthesis (HLS) to generate Verilog implementation from the C code [38]. To ensure that the generated designs are well optimized, we tuned the HLS flow by applying HLS pragmas and targeted the operations that are significantly different between FPGAs and programmable cores (`exp`, `log`, and `sigmoid` functions).

Once the Verilog code is available, we use Yosys 0.5 [40] and Verilog-To-Routing (VTR) 7.0 [13] as our logic synthesis tool chain. Yosys provides support for Verilog-2005. We use the same tools to synthesize, pack, place, and route the accelerator designs on both conventional FPGA and DRAF to ensure fairness. In addition to mapping the designs on reconfigurable architectures, VTR also reports design metrics, including block utilization, critical path delay, and overall area and power consumption, based on low-level component models (discussed below). Related work has showed that the maximum frequencies reported by VTR are about 2x lower than highly-optimized commercial tools [13], [48].

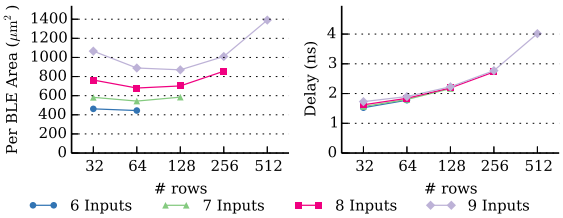


Figure 6. Impact of subarray structure on LUT area and latency: 6 to 9 input bits, 2 output bits, and 8 contexts.

Hence the DRAF results are conservative compared to the programmable core baseline, but the relative performance between DRAF and FPGA remains accurate.

System models: We assume a 45 nm technology process for both baseline FPGA and DRAF. The FPGA is modeled based on Xilinx Virtex-6 devices. We extract the timing and area parameters for each block type from the device datasheet [28] and public literature [32], [48]. Power numbers are extracted from Xilinx Power Estimator (XPE) [49].

For DRAF, we use CACTI-3DD [50] to model the DRAM subarrays. CACTI-3DD models the actual physical structures in commodity DRAM chips using hierarchical wordlines and datalines spanning the array to optimize area [23]. We exclude the global and local bus structures, and only use the subarray model for DRAF LUT. We model the other auxiliary components (FFs, MUXs) in the DRAF BLE similarly to those in FPGAs [32], and rely on VTR for the interconnect fabric modeling.

VI. EVALUATION

A. DRAF Design Exploration

As discussed in Section II-C, a key challenge for DRAF is sizing the DRAM LUT, including the number of input/output bits, the number of contexts, and the exact structure of the DRAM subarray. Unlike conventional DRAM that optimizes mostly for cost, a reconfigurable fabric should be optimized for both performance and cost for its target applications.

We start by tuning the physical subarray in the LUT. Figure 6 shows the area and latency tradeoff when using DRAM subarrays with different numbers of rows and columns. Due to space constraints, we only show results for a LUT with 6 to 9 input bits, 2 output bits (fracturable to 4 bits), and 8 contexts. The number of columns is automatically determined when the numbers of inputs and rows are set. For a fixed number of inputs, as the number of rows increases, the row logic becomes larger while the column logic area decreases. The smallest area is usually with 1- or 2-bit column address. Furthermore, as the number of rows increases, the LUT delay (ACT plus the maximum of PRE and RST, see Figure 4) increases dramatically, primarily due to the dominant bitline latency. This suggests that we should use no more than 128 rows for the LUT.

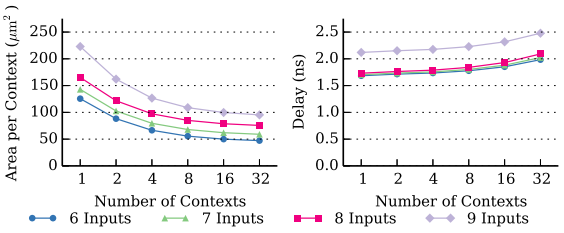


Figure 7. Impact of context number on LUT area and latency. With 6 to 9 input bits, 2 output bits.

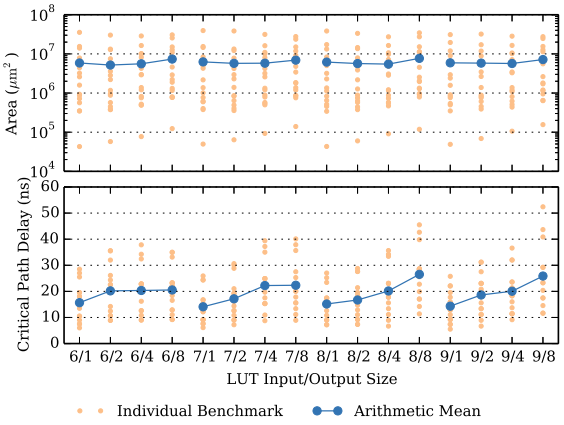


Figure 8. Accelerator design area and critical path latency comparison for different LUT input/output sizes with 8 contexts.

The area and latency impact of the number of contexts (i.e., number of MATs in the subarray) is shown in Figure 7. As the MATs share the row decoder, having more MATs amortizes the large row logic area, resulting in smaller area per context. With more than 8 contexts, the area savings flatten as the MAT area itself dominates. On the other hand, the delay increase is insignificant, as the local wordline drivers isolate the MATs from the master wordline [24], [25]. The 9-input LUT exhibits higher delays because it uses 128 rows while other LUT sizes can use 64 rows (see Figure 6). Overall, 8 to 16 contexts is a good compromise between area efficiency and performance of each context.

Finally, we determine the number of LUT input/output bits. Larger LUTs use larger subarrays which have better area efficiency, but their latencies are longer. Moreover, accelerator designs may not be able to fully utilize all input/output bits of larger LUTs, wasting area. Figure 8 shows the total area and critical path latency for each accelerator design, as well as their arithmetic means, when using different LUT input/output sizes. For area, small designs hardly benefit from the larger LUT size so the number of LUTs needed remains the same, causing the total area to increase. Large designs exhibit slight area savings with larger LUTs. The critical path delays increase for all circuits

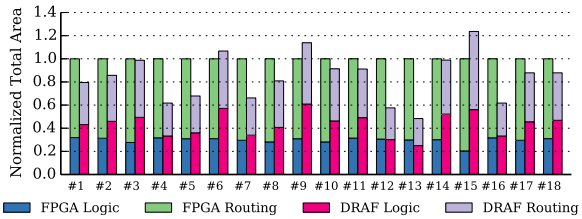


Figure 9. Total area for the accelerator designs on FPGA and DRAF.

when the LUT size increases, mainly due to the use of a larger DRAM subarray. Overall, 7 to 8 inputs with 1 or 2 outputs represent a sweet spot. Unless otherwise specified, in the rest of the paper, we evaluate DRAF with 7-input, 2-output LUTs (fracturable to 4-bit outputs), and 8 contexts. This requires a 2048-bit DRAM subarray for each LUT.

It is worth noting that even though the designs we map to DRAF generally use coarse-grained arithmetic operations, the wider LUT outputs have minimal benefits. This is likely because the current packing algorithm in VTR is not efficient with wide data types. Improving the CAD flow may motivate larger LUTs for DRAF, further improving efficiency.

B. DRAF vs. FPGA

Using the 7-input, 2-output, 8-context LUT for DRAF, we now compare its overall area, performance, and power consumption for all accelerator designs against FPGA. We assume the same ratio of DSP blocks and BRAM blocks in FPGA and DRAF, which are similar to the numbers used in Virtex-6 FPGAs [28]. We map each accelerator to just one of the 8 available contexts in DRAF, and leave the other 7 contexts unused. While unused, the other 7 contexts still contribute to the area, consume leakage power, and introduce a slight access latency penalty in the DRAF LUT. The number of routing tracks used in the arrays is the minimum needed, plus 30% margin which reduces the routing pressure for a better critical path delay [13]. As discussed in Section III-D, the number of routing tracks needed for DRAF is smaller, and for our benchmarks DRAF only requires about half of the tracks of an FPGA.

Figure 9 shows the normalized minimum bounding box area for each benchmark on the two arrays. On average, the area of DRAF is 19% less than that of the FPGA. Note that this means that an 8-context DRAM fabric occupies slightly less area than a single-context FPGA. For the same silicon cost, DRAF can pack 8 designs within the area budget of a single design for an FPGA. This cost advantage makes DRAF a promising shared fabric for accelerator designs.

To compare performance, we show in Figure 10 the synthesized minimum clock period for each accelerator on the FPGA and DRAF. Since the same Verilog design is mapped to both fabrics, the clock period ratio is the ratio of throughput and latency of the accelerator using the two

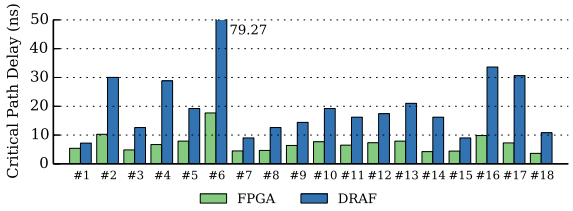


Figure 10. Critical path delay (clock period) for the accelerator designs on FPGA and DRAF.

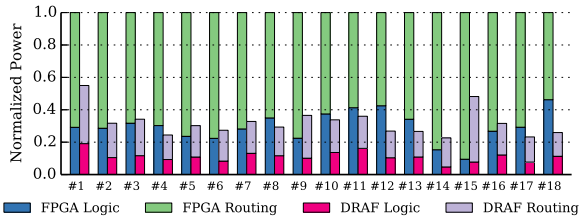


Figure 11. Power consumption for accelerators on FPGA and DRAF.

fabrics. Due to the slow DRAM cell access, DRAF has a 2.74x slower clock frequency than FPGA on average. This slowdown limits the performance gain for DRAF, but as we will see in Section VI-C, DRAF can still provide more than 13x speedup over programmable processors. Benchmark #6 (gmm) has a particularly long critical path in both FPGA and DRAF, because it requires `log` and `exp` operations [7], which VTR currently does not support in the DSP blocks.

Finally, we compare the power consumption of FPGA and DRAF arrays in Figure 11. The FPGA power is dominated by the routing fabric, especially for large designs. DRAF not only reduces the power consumption of logic blocks by using more efficient larger DRAM subarrays as LUTs, but also saves routing power as it requires fewer routing tracks due to denser packing. DRAF's power consumption is only 31% of the FPGA on average. Combined with the performance results, the energy of DRAF is 15% lower than that of an FPGA. Overall, compared to FPGA, DRAF trades off a factor of 2-3x in performance for a 10x improvement in cost (area) and a 3x improvement in power consumption.

C. System Comparison

The fact that DRAF provides power and cost advantages over an FPGA fabric is only significant if it can also provide acceleration over programmable cores. To establish that, we compare the FPGA and DRAF accelerator designs to optimized code running on a single- or multi-core system. We use the C/C++ programs in MachSuite and Sirius and the corresponding OpenCV implementation of the video algorithms for Vivado HLS Video Library. This allows a comparison using the first 13 accelerators in Table II. We run these programs on a single core of a 2.3 GHz Intel Xeon

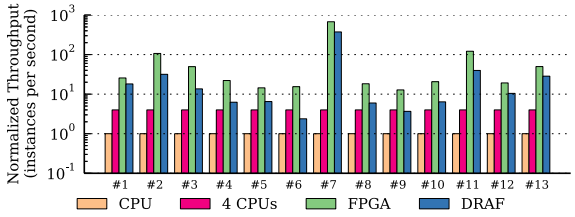


Figure 12. Performance comparison between single-core, multi-core, FPGA, and DRAF.

E5-2630 processor. We optimistically assume that a 4-core implementation would achieve perfect speedup of 4 over the single core results by exploiting request-level parallelism in datacenter workloads. The die area for the FPGA and DRAF devices is assumed to be 75 mm^2, and we fit as many instances of the accelerators as the area allows us to.

Figure 12 compares performance between the four platforms. Both the FPGA and DRAF outperform a single core for all accelerator tasks. The average speedup for the FPGA is 37.13x and for DRAF is 13.45x. #7 harris shows drastic speedup (676x for FPGA and 374x for DRAF) due to the efficient use of hardware line buffers for the image kernels. Even if we assume perfect speedup with 4 cores, FPGA and DRAF still exhibit significant improvements (9.28x for FPGA and 3.36 for DRAF). DRAF is outperformed by the 4-core design for two accelerators: #6 (gmm) is 1.68x slower due to the use of `exp` and `log` (see Figure 10); #9 (mergesort) is only 8% slower since the in-place sorting is bounded by the number of BRAM memory ports.

To understand how power consumption compares across platforms, we estimate the maximum power consumption for an FPGA and a DRAF device with 75 mm^2 die size. We assume full resource utilization for both devices. The FPGA clock frequency is set to 400 MHz and 135 MHz for DRAF. Under these conditions, the FPGA power consumption is 15.89 W, while DRAF consumes just 3.57 W. The normal-case operation power is even smaller. For instance, with the benchmarks above, the average FPGA power is 2.16 W and DRAF power is 0.63 W. Hence, DRAF-based accelerators of various sizes can be easily added to servers without significantly affecting their power budgets. In comparison, a single core in a Xeon-class server chip normally consumes 7 to 10 W with TDP as high as 15 W [51], and a 4-core solution would consume 4 times more.

In terms of thermal issues, DRAF's power consumption is higher than that of conventional DRAM chips, which is usually about 0.5 W (based on I_{DD7}). This can potentially increase the die temperature, and may require higher refresh rate for DRAM arrays. However, just as in the case of FPGAs, we can apply an active heat sink to help keep the temperature within 85 °C. XPE suggests even with the maximum traditional FPGA power density a normal fan can

maintain the temperature at 72 °C [49], thus a low-end fan is sufficient for DRAF, given its much lower power.

VII. Related Work

FPGAs as datacenter accelerators: FPGAs are becoming popular as accelerators for important online and analytics services. Microsoft designed large-scale, reconfigurable fabric where each application can use FPGAs across multiple servers connected using dedicated interconnect [6]. This platform has been used thus far to accelerate websearch and deep neural network processing [52]. Baidu proposed an FPGA-based accelerator for large-scale deep neural networks [9]. IBM recently developed a reconfigurable platform for fast text analytics [8]. Convey and Micron used their hybrid threading toolset to accelerate image processing and memcached with FPGAs [53]. In addition to industrial designs, numerous FPGA-based accelerators for datacenter workloads have been proposed by research efforts [7], [47], [54], [55]. DRAF offers a high density (low cost), low power, reconfigurable fabric for such datacenter accelerators.

Enhancing FPGA reconfigurability: Several techniques have been proposed to enhance FPGAs, including coarse-grained blocks like DSP units which are already in use. Coarse-Grained Reconfigurable Arrays (CGRAs) deviate from bit-level reconfigurability and use a large number of wide functional units as their basic elements [56]–[59]. CGRAs provide good efficiency for analytics applications that mostly perform wide arithmetic operations. The current design for DRAF focuses on fine-grained programmability to cover a wider set of workloads.

Among prior multi-context FPGAs [16]–[18], DPGA used DRAM technology but only as the backup configuration storage [16]. In contrast, DRAF uses DRAM subarrays directly as LUTs, avoiding the energy and latency overheads of switching contexts. DPGA also used 3T DRAM cells and smaller DRAM arrays, while DRAF uses the 1T cells and dense subarrays common in commercial DRAM.

As FPGAs grow in size, the interconnect complexity increases significantly. Dynamic packet-switched and time-multiplexing routing protocols have been explored to replace the fully-static routing schemes in modern FPGAs [14], [15], [36], [37]. While dynamic routing leads to modest area savings, it introduces high latency and power overheads and requires complicated scheduling algorithms. DRAF provides significant area and power savings at the device level. Hence the routing complexity of DRAF devices is similar to a moderate sized FPGAs; dynamic routing is not necessary.

The closest proposal to DRAF is the Micron's Automata Processor (AP) [60]. AP uses DRAM technology to realize a reconfigurable architecture, but does so quite differently from DRAF. First, AP has a different and limited programming model, where applications must be expressed as an instance of automata processing, which is difficult for general algorithms. Second, AP is state-machine-oriented; while this is extremely efficient for large-scale state-machines, such as regular expressions [60], the general logic required for many of the applications we consider is difficult. AP also does not provide multi-context support, lowering the capabilities of the device in terms of reconfiguration.

DRAM optimizations: There is significant work on improving the latency, throughput, area, and energy of DRAM. The hierarchical array structures have been historically employed in DRAM chips to reduce access latency [24], [25], and were recently leveraged to also reduce energy consumption for multi-core applications [23]. The subarray size introduces a well-known tradeoff between latency and area. CHARM [26] and tiered-latency DRAM [27] leveraged heterogeneity to provide different sizes of subarrays in the same DRAM chip to achieve the best of both worlds. Others have also modified the subarray organization to expose subarray-level parallelism (SALP) and increase throughput [20]. DRAF builds upon these techniques to implement an efficient LUT with DRAM. Our focus in DRAM engineering is mostly on delay and power consumption, instead of area, which leads us to slightly different design decisions.

Processing-In-Memory or Near-Data Processing: PIM and NDP proposed integrating processing logic and DRAM arrays in the same silicon die [43], [61]–[63] or through 3D stacking [44], [64]–[66]. Some of these proposals used reconfigurable logic [43], [44]. However, DRAM was exclusively used for data storage. In DRAF, DRAM arrays *are* the processing elements. We refer to this pattern as *Processing-Using-Memory* in contrast to PIM and NDP.

VIII. Conclusion

This paper proposed DRAF, a bit-level reconfigurable fabric based on dense DRAM subarrays. DRAF trades off some of the performance of an FPGA for significant improvements in area density and power consumption. It also provides support for multiple contexts, allowing multiple applications and tasks to share a DRAF device for multiple accelerators without significant context switch overheads. We demonstrate that DRAF provides a 10x area density and a 3.2x power improvement over FPGA. A modestly-sized DRAF device in terms of area and power provides a 13x speedup and a 11x power improvement over a programmable core for acceleration tasks common to datacenter workloads. These features make DRAF a great reconfigurable substrate for acceleration for power and cost constrained environments, where a large number of accelerators may be necessary across a wide range of workloads.

Acknowledgment

The authors want to thank the anonymous reviewers for their insightful comments. This work was supported by the Stanford Pervasive Parallelism Lab, the Stanford Platform Lab, the Center for Future Architectures Research (C-FAR), Samsung, and NSF grant SHF-1408911.

REFERENCES

[1] H. Esmaeilzadeh *et al.*, "Dark silicon and the end of multicore scaling," in *ISCA '11*, 2011.

[2] M. Horowitz, "Computing's energy problem (and what we can do about it)," in *ISSCC '14*, 2014.

[3] M. B. Taylor, "Is dark silicon useful? harnessing the four horesemen of the coming dark silicon apocalypse," in *DAC '12*, 2012.

[4] R. Tessier, K. Pocek, and A. DeHon, "Reconfigurable Computing Architectures," *Proceedings of the IEEE*, vol. 103, March 2015.

[5] A. DeHon, "Fundamental Underpinnings of Reconfigurable Computing Architectures," *Proceedings of the IEEE*, vol. 103, March 2015.

[6] A. Putnam *et al.*, "A Reconfigurable Fabric for Accelerating Large-scale Datacenter Services," in *ISCA '14*, 2014.

[7] J. Hauswald *et al.*, "Sirius: An Open End-to-End Voice and Vision Personal Assistant and Its Implications for Future Warehouse Scale Computers," in *ASPLOS '15*, 2015.

[8] R. Polig *et al.*, "Giving Text Analytics a Boost," *Micro, IEEE*, vol. 34, no. 4, pp. 6–14, July 2014.

[9] J. Ouyang, "SDA: Software-Defined Accelerator for Large-Scale DNN Systems," in *HotChips*, 2014.

[10] S. M. Trimberger, "Three Ages of FPGAs: A Retrospective on the First Thirty Years of FPGA Technology," *Proceedings of the IEEE*, vol. 103, no. 3, pp. 318–331, March 2015.

[11] U. Hölzle and L. A. Barroso, *The Datacenter As a Computer: An Introduction to the Design of Warehouse-Scale Machines*, 1st ed. Morgan and Claypool Publishers, 2009.

[12] I. Kuon, R. Tessier, and J. Rose, "FPGA Architecture: Survey and Challenges," *Foundations and Trends in Electronic Design Automation*, 2008.

[13] J. Luu *et al.*, "VTR 7.0: Next Generation Architecture and CAD System for FPGAs," vol. 7, no. 2, June 2014, pp. 6:1–6:30.

[14] B. Van Essen *et al.*, "Static versus scheduled interconnect in Coarse-Grained Reconfigurable Arrays," in *FPL '09*, 2009.

[15] N. Kapre *et al.*, "Packet Switched vs. Time Multiplexed FPGA Overlay Networks," in *FCCM '06*, 2006.

[16] E. Tau *et al.*, "A First Generation DPGA Implementation," in *FPD*, 1995.

[17] S. Trimberger *et al.*, "A time-multiplexed FPGA," in *FCCM*, 1997.

[18] T. R. Halfhill, "Tabula's Time Machine," *Microprocessor Report*, vol. 131, 2010.

[19] JEDEC Standard, "DDR3 SDRAM Standard," JESD79-3, 2012.

[20] Y. Kim *et al.*, "A Case for Exploiting Subarray-level Parallelism (SALP) in DRAM," in *ISCA '12*, 2012.

[21] T. Yamauchi *et al.*, "The Hierarchical Multi-Bank DRAM: A High-Performance Architecture for Memory Integrated with Processors," in *ARVLSI '97*, 1997.

[22] T. Vogelsang, "Understanding the Energy Consumption of Dynamic Random Access Memories," in *MICRO '43*, 2010.

[23] A. N. Udipi *et al.*, "Rethinking DRAM Design and Organization for Energy-constrained Multi-cores," in *ISCA '10*, 2010.

[24] M. Nakamura *et al.*, "A 29 ns 64 Mb DRAM with Hierarchical Array Architecture," in *ISSCC '95*, 1995.

[25] Y. Nitta *et al.*, "A 1.6 GB/s Data-Rate 1 Gb Synchronous DRAM with Hierarchical Square-Shaped Memory Block and Distributed Bank Architecture," in *ISSCC '96*, 1996.

[26] Y. H. Son *et al.*, "Reducing Memory Access Latency with Asymmetric DRAM Bank Organizations," in *ISCA '13*, 2013.

[27] D. Lee *et al.*, "Tiered-latency DRAM: A Low Latency and Low Cost DRAM Architecture," in *HPCA '13*, 2013.

[28] Xilinx Inc., "DS152: Virtex-6 FPGA Data Sheet: DC and Switching Characteristics," March 2014, v3.6.

[29] Altera Corporation, "Stratix Series FPGAs & SoCs," https://www.altera.com/products/fpga/stratix-series.html.

[30] Xilinx Inc., "Virtex-7 FPGAs Products," http://www.xilinx.com/products/silicon-devices/fpga/virtex-7.html.

[31] A. DeHon, "DPGA Utilization and Application," in *FPGA*, 1996.

[32] V. Betz *et al.*, Eds., *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, 1999.

[33] R. Balasubramonian, M. Shevgoor, and J.-S. Kim, "A DRAM Refresh Tutorial," http://utaharch.blogspot.com/2013/11/a-dram-refresh-tutorial.html, November 2013.

[34] X. Dong and J. Suh, "DRAM sub-array level refresh," March 2015, uS Patent 8,982,654.

[35] M. J. Lyons *et al.*, "The Accelerator Store: A Shared Memory Framework for Accelerator-based Systems," *TACO*, vol. 8, no. 4, pp. 48:1–48:22, Jan. 2012.

[36] R. Gindin, I. Cidon, and I. Keidar, "NoC-Based FPGA: Architecture and Routing," in *NOCS*, 2007.

[37] R. Francis and S. Moore, "Exploring Hard and Soft Networks-on-Chip for FPGAs," in *FPT*, 2008.

[38] Xilinx Inc., "Vivado High-Level Synthesis (HLS)," http://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html, 2014.

[39] Altera Corporation, "Altera SDK for Open Computing Language (OpenCL)," https://www.altera.com/products/design-software/embedded-software-developers/opencl/overview.html.

[40] C. Wolf, "Yosys Open Synthesis Suite," http://www.clifford.at/yosys/.

[41] Xilinx Inc., "UltraScale Architecture and Product Overview," October 2015, v2.4.

[42] R. Balasubramonian *et al.*, "Near-Data Processing: Insights from a MICRO-46 Workshop," *Micro, IEEE*, July 2014.

[43] M. Oskin, F. T. Chong, and T. Sherwood, "Active Pages: A Computation Model for Intelligent Memory," in *ISCA*, 1998.

[44] M. Gao and C. Kozyrakis, "HRL: Efficient and Flexible Reconfigurable Logic for Near-Data Processing," in *HPCA*, 2016.

[45] J. Hamilton, "Internet-scale service infrastructure efficiency," in *ISCA*, 2009.

[46] B. Reagen *et al.*, "MachSuite: Benchmarks for accelerator design and customized architectures," in *IISWC*, 2014.

[47] V. K. Kundeti, "Synthesizable, Space and Time Efficient Algorithms for String Editing Problem," Master's thesis, University of Connecticut, 2008.

[48] E. Hung, F. Eslami, and S. Wilton, "Escaping the Academic Sandbox: Realizing VPR Circuits on Xilinx Devices," in *FCCM*, 2013.

[49] Xilinx Inc., "Xilinx Power Estimator (XPE)," http://www.xilinx.com/products/technology/power/xpe.html, 2012, v14.3.

[50] K. Chen *et al.*, "CACTI-3DD: Architecture-Level Modeling for 3D Die-stacked DRAM Main Memory," in *DATE '12*, 2012.

[51] Intel, "Measuring Processor Power: TDP vs. ACP," http://www.intel.com/content/dam/doc/white-paper/resources-xeon-measuring-processor-power-paper.pdf, 2011.

[52] K. Ovtcharov *et al.*, "Accelerating Deep Convolutional Neural Networks Using Specialized Hardware," White paper, February 2015.

[53] T. Brewer, "Convey's Acceleration of the Memcached and Imagemagick Applications," in *CARL*, 2015.

[54] E. C. Lin *et al.*, "A 1000-word Vocabulary, Speaker-independent, Continuous Live-mode Speech Recognizer Implemented in a Single FPGA," in *FPGA*, 2007.

[55] Y. Sun *et al.*, "Accelerating Frequent Item Counting with FPGA," in *FPGA*, 2014.

[56] K. Choi, "Coarse-Grained Reconfigurable Array: Architecture and Application Mapping," *IPSJ*, vol. 4, pp. 31–46, 2011.

[57] B. Mei *et al.*, "ADRES: An Architecture with Tightly Coupled VLIW Processor and Coarse-Grained Reconfigurable Matrix," in *FPL*, 2003.

[58] H. Singh *et al.*, "MorphoSys: An Integrated Reconfigurable System for Data-Parallel and Computation-Intensive Applications," *IEEE Trans. Comput.*, vol. 49, no. 5, pp. 465–481, May 2000.

[59] V. Govindaraju *et al.*, "Dynamically Specialized Datapaths for Energy Efficient Computing," in *HPCA*, 2011.

[60] P. Dlugosch *et al.*, "An Efficient and Scalable Semiconductor Architecture for Parallel Automata Processing," *TPDS*, Dec 2014.

[61] D. Patterson *et al.*, "A Case for Intelligent RAM," *Micro, IEEE*, vol. 17, no. 2, pp. 34–44, Mar 1997.

[62] M. Hall *et al.*, "Mapping Irregular Applications to DIVA, a PIM-based Data-Intensive Architecture," in *SC*, 1999.

[63] Y. Kang *et al.*, "FlexRAM: Toward an Advanced Intelligent Memory System," in *ICCD*, 2012.

[64] S. Pugsley *et al.*, "NDC: Analyzing the Impact of 3D-Stacked Memory+Logic Devices on MapReduce Workloads," in *ISPASS*, 2014.

[65] J. Ahn *et al.*, "A Scalable Processing-in-memory Accelerator for Parallel Graph Processing," in *ISCA*, 2015.

[66] M. Gao, G. Ayers, and C. Kozyrakis, "Practical Near-Data Processing for In-memory Analytics Frameworks," in *PACT*, 2015.